Bachelor's Thesis

Information and Communications Technology

2020

Lyudmila Grigoryeva

IMPLEMENTATION OF LOG MANAGEMENT SOLUTION IN A MEDICINE DISPENSER ROBOT

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2020 | 43, 2

Lyudmila Grigoryeva

IMPLEMENTATION OF LOG MANAGEMENT SOLUTION IN A MEDICINE DISPENSER ROBOT

System logs play a crucial role when it comes to tracking issues, monitoring data and diagnosing problems within a system. Eventually, the amount of log data generated by systems and devices exceeds the amount that can be managed manually, and companies start looking for a proper log management solution.

One of the goals of this thesis was to study log management and the necessary steps that need to be taken before log data can be used for further analysis. One of such steps is data preprocessing, the process of removing inconsistencies and errors in raw data. During this study, a data preprocessing prototype was built to transform the case company log files that contained unformatted and incorrect timestamps.

Another goal was to research the Elastic Stack and its components to determine how they meet the basic functionalities of a log management solution. This particular solution was chosen based on it being open-source, lightweight and flexible. The theory is backed by an Elastic Stack implementation, and the thesis supports the initial claim that the Elastic Stack is a suitable choice for a log management solution.

KEYWORDS:

Data preprocessing, Log Management, Elastic Stack

TABLE OF CONTENTS

APPENDICES

PICTURES

TABLES

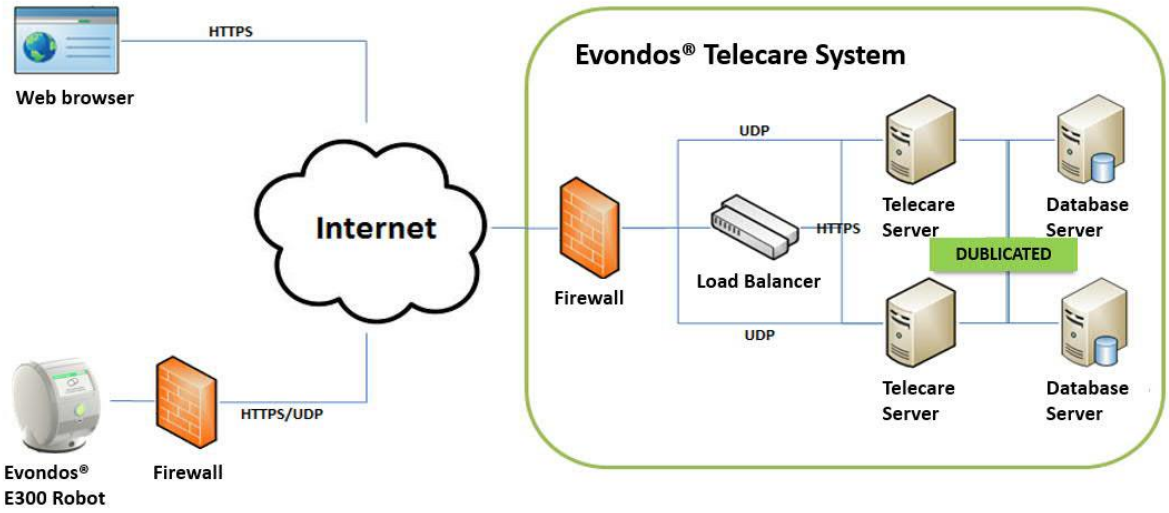
LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-separated values
ISO	International Organization for Standardization
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
RAM	Random-access memory
SSD	Solid-state drive
XML	Extensible Markup Language

1 INTRODUCTION

This thesis was commissioned by Evondos Ltd, a Finnish healthcare service company based in Salo. Outside of Finland, Evondos operates sales offices in Sweden, Norway and Denmark.

Evondos provides a digital health solution that addresses the issues in medicine distribution to patients. The system includes the Evondos® E300 Medicine Dispensing Robot and the Evondos® Telecare System (Picture 1):



Picture 1. Evondos service system architecture. (Evondos, 2019)

The Evondos system provides automatic medicine dispensation to patients with chronic conditions and long-term medication prescriptions. It supports patients' well-being and quality of life by providing an effective service that:

- Ensures correct medication is delivered at the right time with the right dosage
- Ensures drug safety through remote monitoring of medication
- Reduces the need for routine home calls regarding taken medication
- Alleviates the pressure on care staff workers dealing with manual medication distribution

(Evondos, 2019)

The thesis was carried out for Evondos with the purpose of preparing a technical solution that will help the company set up a suitable and working log management solution.

The goals of the study were defined as to:

1. Investigate the process of log management
2. Determine how data should be preprocessed before being ingested into a log management system
3. Implement a log management solution

The first chapter of the thesis concentrates on the importance of logging, and explains what are log management and log analysis. The second chapter takes a look at data preprocessing and the different types of data wrangling that are performed during this step. The chapter focuses specifically on data cleansing and how it can be done through scripting. The third chapter of the thesis defines the important features of log management solutions that need to be considered when choosing a particular solution. The fourth chapter provides the theoretical background of the Elastic stack, which is implemented in section five. Chapter six concludes this work by summarizing what has been studied and achieved during this study.

2 BACKGROUND

2.1 Log files and logging

Log files are file extensions that store all the events that occur in a system or when running some software. These files consist of entries, where each entry contains information regarding what happens after the system is started and until it is shut down. The process of recording this information is known as logging. The purpose of logging is to keep track of all the events within a system in order to understand what is going on in the system and diagnose potential issues such as running out of disk space and many others. (Shields, 2017)

Log files are highly configurable and, generally, there are no strict rules regarding the size, format or location of the log. Different systems and applications use various approaches when it comes to logging and organizing log files. Some will have combined log file entries from different sources in one file, while others can have separate files for error and access logs. (Solarwinds Loggly, n.d.) However, in order to avoid software developers having to design their own ad hoc logging systems, some standard logging formats were introduced, such as the Common Log Format for server logs and syslog for Unix-based systems.

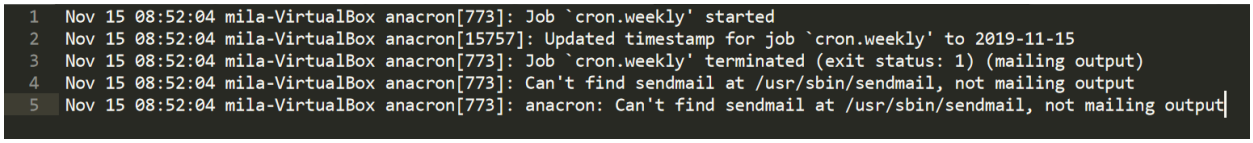
Although, as mentioned before, there are no standards when it comes to logging, most of log files contain the following fields (Picture 2, Picture 3):

1. Timestamp
2. Category or event classification
3. Descriptive message

(La rosa, 2018)

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
```

Picture 2. Common Log Format.



Picture 3. Syslog messages.

2.2 Log management

Generating and configuring log files can become problematic and less valuable when proper log management is missing. Log management is an important procedure that deals with the vast amount of log files that are being produced. Generally, log management includes the following:

1. Log generation
2. Log storage
3. Log analysis
4. Alerting and notification

(Kent & Murugiah, 2006)

2.2.1 Log generation and storage

Log generation involves the process of generating log files as well as determining the sources that perform logging. If multiple sources are responsible for generating log files, it is important to ensure the consistency between the different log files.

Log storage, depending on the requirements of the system, can become complicated and tricky. This part of log management requires determining how much log data should be stored and where. Log files can be stored locally and also sent to one or more servers. However, local storage is not always possible due to limited space in the log generator, so, often, logs are sent to a remote server. Storing log entries in a database is another option, which provides a storage format that can be helpful during log analysis. (Kent & Murugiah, 2006)

Log storage also involves log rotation, which is the process of closing a log file and opening the next one once the previous log file is considered complete. Basically log rotation determines how much data a single log file should contain. Logs can be rotated on a regular timely basis, e.g. hourly, daily, weekly or when the log files reach a certain

size, e.g. 1 megabyte, 10 megabytes, 100 megabytes. Log rotation can be implemented in the log generator or configured with a third-party utility. (Kent & Murugiah, 2006)

Another part of log storage is log normalization, which is the process of converting data to a single format. Log normalization ensures data analysis can be performed at ease, especially when there are multiple log sources in the system. Normalization, however, can be resource-consuming. (Kent & Murugiah, 2006)

Before logs are sent to a remote server, system administrators and software developers should also need to determine if log files need to be encrypted and if certain information needs to be protected. (Kent & Murugiah, 2006)

2.2.2 Log analysis

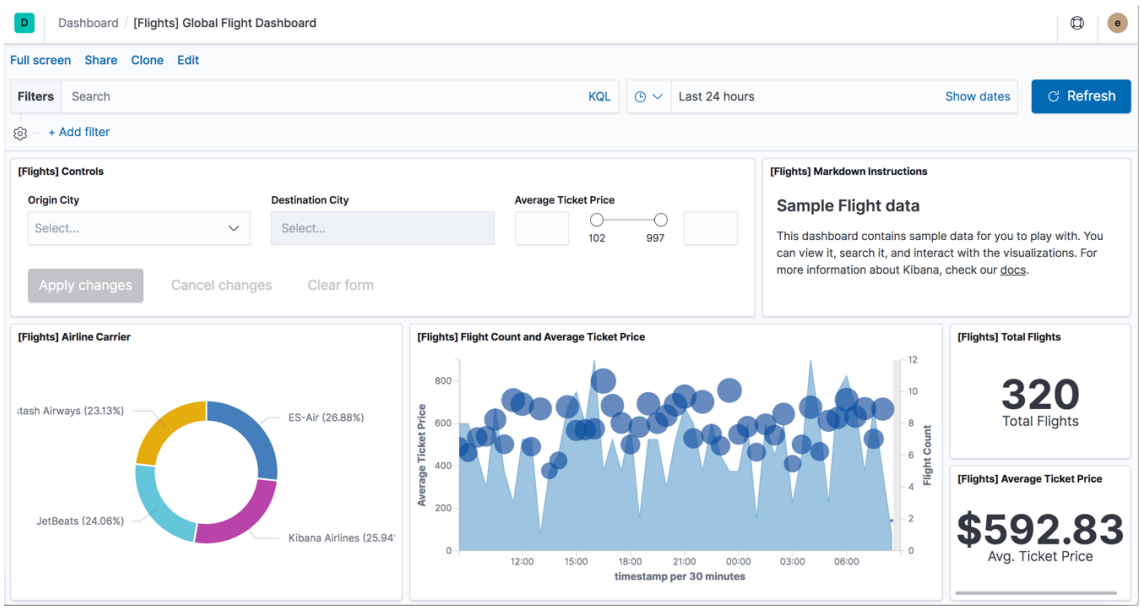
Log analysis is the process of reading, reviewing and studying log files. Log analysis helps developers and system administrators to troubleshoot issues in the OS or debug software applications. (Sumo Logic, n.d.)

Once it has been determined how log data is stored and collected from log generators, log analysis can take place. Traditionally, log files have been configured without built-in analysis tools. Nowadays, some log generators might provide limited analysis capabilities. Before log data can be analyzed, log files also need to be centralized if they are coming from multiple log generating sources. (Kent & Murugiah, 2006)

Log analysis can be performed using various functions and methods. The most common way to access log data across various sources is through searching. When building a log management system one of the important things is to set up a clean and responsive search interface. (Chuvakin, 2010) It can be implemented using a range of tools: from simple command-line utilities such as *grep* to third-party software that are powered by machine learning. Those machine learning algorithms can even provide predictive analysis beyond detecting security errors and solving everyday issues.

Indexing is one of the key components of a reliable and successful log management system that optimizes its performance (sometimes by a factor of a hundred in terms of speed). Indexing is done by creating an index, a data structure that allows for quick search queries across the log storage. Not all log management systems, however, support indexing. (Chuvakin, 2010)

One way to review data without diving too deep into log analysis and getting caught up in endless search queries is visualization. Visualizing logs gives the possibility to take a quick glance at issues and events in the system as well as its performance and availability through different graphs and dashboards. (Nguyen, 2018)



Picture 4. Kibana visualization dashboard. (Elastic.co, 2019e)

2.2.3 Alerting and notification

At the beginning of this chapter, logging was defined as an important process used for diagnosing issues and getting a better understanding of what is going on in the system. Parsing and indexing log files helps developers perform quick searches across thousands of lines of their log data. In order to avoid constant searching, a good option would be to set up some alerting and monitoring tool to watch out for specific events and trigger some kind of endpoint such as e-mail, alarm or Slack channel. (Solarwinds Loggly, 2019)

3 DATA PREPROCESSING

3.1 Preprocessing

Before any data can be processed and used for analysis, it is important to ensure the high quality of data in order to get the best results and predictions from the analysis. Some attributes that define data quality:

1. Accuracy

   Accuracy in data refers to the amount of erroneous values in the results. Inaccurate data can occur in datasets due to human or computer error as well as someone deliberately adding incorrect or false data. Incorrect data formats and duplicate values can also lead to inaccuracy in a dataset.

2. Completeness

   Data is incomplete when some fields and attributes are missing. This can be caused by accidental deletion of existing data or simply because data was not available at the time when it was collected.

3. Consistency

   When data originates from multiple sources, it is important it is aggregated consistently.

(Mushtaq, 2019)

In theory, log generation should be configured in such way that it would ensure data consistency and accuracy from the start. However, real-world data is often incomplete, contains errors and discrepancies and is not centralized due to reasons described above. This is when preprocessing comes into the picture.

During preprocessing, log files that contain raw data can be parsed, cleaned, normalized and transformed. (Sharma, 2018) Depending on what is compromising the quality of the dataset, a variety of actions can be performed upon the dataset. Those actions are:

1. Data cleaning: replace missing values, correct inconsistencies, remove noise (random variance in data) and outliers

2. Data transformation: normalize data
3. Data reduction: reduce the volume of the dataset without losing the integrity of original for easier data handling
4. Data integration: propagate data and detect conflicting values

(Mushtaq, 2019)

3.2.1 Data cleaning

Based on the kind of data cleaning that needs to be performed, different techniques are applied. Data can be replaced, transformed, removed. Let's take a closer look at how raw data can be transformed in one of the sample log files.

Timestamps identify when a specific event occurs in the system. Sloppy implementations can make it difficult to parse log data if the timestamps are incorrect and inconsistent.

In this particular case, one thing that requires attention is the timestamp format. It is currently non-standard, missing the year attribute and the timezone offset.

```
Oct  3 12:26:30 example.info: This is an example log message…..
```

The example above illustrates a sample timestamp from the log file. It needs to be converted to ISO format, which is a standard way of representing date- and time- related data. (International Organization for Standardization, n.d.) The date and time objects that follow ISO look like this:

```
Date: YYYY-MM-DD or YYYYDDD
Time:  hh:mm:ss or hhmmss
```

For combined time and date representations, the following format is followed:

```
<date>T<time>
```

Time zones in ISO can be represented as local time, UTC (+00:00), or an offset from UTC, which is the difference in hours from Coordinated Universal Time.
The sample timestamp following the ISO format would look like this due to the missing year attribute:

```
--10-03
```

Preprocessing, or, more specifically, data cleaning, will add the year attribute to the timestamp.

For now, it will be assumed that the log file is created in 2019 and the device is located in Finland. Hence, the timestamp will need to look like:

```
2019-10-03T12:26:30+02:00
```

Python will be the chosen programming language for wrangling the timestamps in the log files. To do that, the following modules need to be imported: *datetime*, *pytz*, *re*:

- datetime provides classes for manipulating date and time objects.
- pytz provides a database of time zones
- re provides regular expression matching

(Python Software Foundation, 2019)

Regular expressions can be used to extract timestamps from the message line:

```
date_match = re.search(r'\w\w\w \s?\d{1,2} ', line)
time_match = re.search(r'\d{2}:\d{2}:\d{2}', line)
raw_timestamp = date_match.group() + year + " " + time_match.group()
```

The function *strptime* takes a given string and formats it to a Python datetime object:

```
timestamp = datetime.datetime.strptime(raw_timestamp, '%b %d %Y %H:%M:%S')
```

```
>>> 2019-10-03 12:26:30
```

Now the resulting timestamp is a datetime object, but is still represented as “native” time without any time zone information. To fix that the pytz module imported earlier can be used to define the time zone the device is in, in this case, the Helsinki time zone.

```
timezone = pytz.timezone("Europe/Helsinki")
aware_timestamp = timezone.localize(timestamp)
```

Now the timestamp is aware of the time zone it is in:

```
>>> 2019-10-03 12:26:30+03:00
```

The last step is to represent the timestamp according to the ISO format. To achieve that, the *isoformat* function from the datetime module can be used:

```
iso_timestamp = aware_timestamp.isoformat()
```

```
>>> 2019-10-03T12:26:30+03:00
```

Another issue that needs to be addressed is the inconsistency between timestamps in log messages that were combined from different sources. Here is an example of such sample:

```
Apr 29 10:23:02 mila-VirtualBox nm-dispatcher: req:1 'down' [enp0s3]: start running
ordered scripts...
Jan  1 02:00:11 mila-VirtualBox anacron[736]: Job `cron.daily' terminated
Jan  1 02:00:17 mila-VirtualBox dhclient[3533]: DHCPACK of 10.0.2.15 from 10.0.2.2
Jan  1 02:00:24 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5180]
address 10.0.2.15
Jan  1 02:00:36 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5188]
plen 24 (255.255.255.0)
Jan  1 02:00:37 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5195]
gateway 10.0.2.2
Jan  1 02:00:43 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5209]
lease time 86400
Jan  1 02:00:48 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5209]
lease time 86400
Jan  1 02:01:12 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5215]
nameserver '192.168.135.11'
Jan  1 02:02:40 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5215]
nameserver '192.168.135.12'
Apr 29 10:26:02 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5215]
nameserver '192.168.135.11'
Apr 29 10:26:02 mila-VirtualBox NetworkManager[734]: <info>  [1574416265.5215]
nameserver '192.168.135.12'
```

Here some of the log messages do not have access to the real-time clock and generate incorrect timestamps. This complicates log analysis if the raw data is not transformed before being ingested into further storage.

In order to extrapolate the new timestamps, the information provided by the incorrect timestamps will be used. It is possible to estimate how much time has roughly passed before time is set to the correct value by looking at the last line with incorrect time:

```
>>> 02:02:40
```

Using this value and the values in each message line, new timestamp values can be extrapolated. First, the correct timestamps should be formatted to a datetime object:

```
>>> ('2019-04-29 10:26:02', '02:02:40')
```

The hour attribute needs to be removed:

```
# correct_time = 2019-04-29 10:26:02
correct_time = datetime.datetime.strptime(correct_time_str, "%Y-%m-%d %H:%M:%S")

# minutes_str = 02:02:40
minutes = datetime.datetime.strptime(minutes_str, "%H:%M:%S").replace(hour=0)
```

By subtracting the amount of minutes passed during the incorrect timestamp phase and adding the time passed in each individual line, the correct representation of the timestamp can be achieved:

```
# old_time_str = incorrect timestamp in the message line
old_time_str = re.search(r'\d{2}:\d{2}:\d{2}', line).group()
old_time = datetime.datetime.strptime(old_time_str, "%H:%M:%S").replace(hour=0)
new_time = correct_time - minutes + old_time
```

By using the previously mentioned functions to format the timestamps to ISO and make sure they are aware of the time zone, this results in:

```
2019-04-29T10:23:02+03:00
2019-04-29T10:23:33+03:00
2019-04-29T10:23:39+03:00
2019-04-29T10:23:46+03:00
2019-04-29T10:23:58+03:00
2019-04-29T10:23:59+03:00
2019-04-29T10:24:05+03:00
2019-04-29T10:24:10+03:00
2019-04-29T10:24:34+03:00
2019-04-29T10:26:02+03:00
2019-04-29T10:26:02+03:00
2019-04-29T10:26:02+03:00
```

The final corrected log sample will now look like:

```
2019-04-29T10:23:02+03:00 mila-VirtualBox nm-dispatcher: req:1 'down' [enp0s3]:
start running ordered scripts...
2019-04-29T10:23:33+03:00 mila-VirtualBox anacron[736]: Job `cron.daily' terminated
2019-04-29T10:23:39+03:00 mila-VirtualBox dhclient[3533]: DHCPACK of 10.0.2.15 from
10.0.2.2
2019-04-29T10:23:46+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5180]   address 10.0.2.15
2019-04-29T10:23:58+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5188]   plen 24 (255.255.255.0)
2019-04-29T10:23:59+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5195]   gateway 10.0.2.2
2019-04-29T10:24:05+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5209]   lease time 86400
2019-04-29T10:24:10+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5209]   lease time 86400
2019-04-29T10:24:34+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5215]   nameserver '192.168.135.11'
2019-04-29T10:26:02+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5215]   nameserver '192.168.135.12'
2019-04-29T10:26:02+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5215]   nameserver '192.168.135.11'
2019-04-29T10:26:02+03:00 mila-VirtualBox NetworkManager[734]: <info>
[1574416265.5215]   nameserver '192.168.135.12'
```

As mentioned before, clean and concise datasets are crucial in log analysis. It is important to understand where the data comes from especially in scenario where data sets have a geographic scope (e.g. devices are located in different time zones while the log files log timestamps as native values).

This example took a closer look at timestamps, and how they can be formatted and cleaned up for further log processing. In order to get a clear picture of what is happening in the log file, one can ask some key questions before parsing the data:

1. How were the log files generated? (Was the information acquired from multiple sources?)
2. Are the timestamps standardized?
3. Are the timestamps aware of the time zones the device or system is located in?
4. How are timestamps created in the log files?
5. If the timestamps are inconsistent, where can you get relatively correct information?

Incorrect information can pose challenges to someone who is trying to perform log analysis. However, proper preparation and understanding the data set you are working with can help you to smoothly preprocess the data. This example demonstrated how to programmatically handle an issue with timestamps in a given log file by using a simple piece of code.

4 ELASTIC STACK

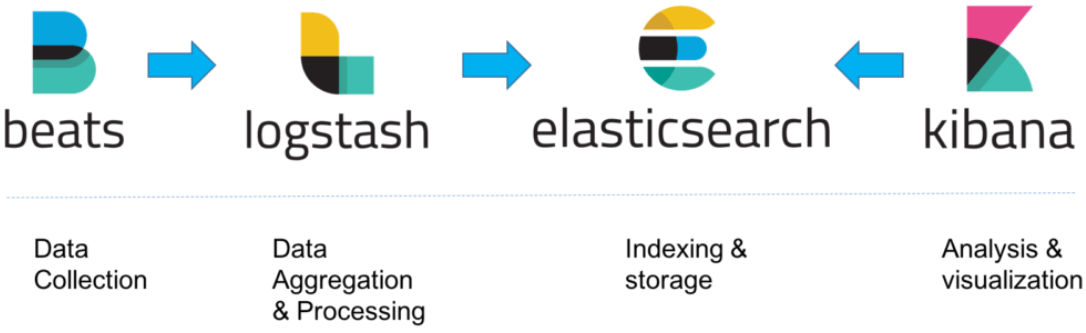
Companies have been using logs to identify potential performance issues and to monitor their systems for a while now. The one thing that has become a challenge when it comes to log management is the amount of data that is constantly being produced. Before software engineers could simply log into their application and use *grep* to search through the log files. Nowadays to perform proper log analysis engineers need to use more advanced and modern log management tools.

Splunk has been the market leader specifically for log management (and handling big data in general) for quite a long time. It offers numerous functionalities for searching and analysing machine-generated big data which can be setup on-premise as well as on the cloud. While Splunk is a mature product that offers a large variety of tools and is used by 12,000 users including companies such as Atlassin, Adobe, Coca-Cola and Porsche, it comes with a high price tag. (Yigal, 2017)

The Elastic Stack, on the other hand, hasn't been around for too long. Yet it is downloaded about 500,000 times every month, and the number keeps growing. Why is it so popular? The Elastic Stack is open source, which most likely explains the popularity. The main benefits of using open-source solutions for IT organizations is being able to avoid vendor lock-in and ensure better security. The latter comes from having access to the source code which might mean issues will get discovered earlier since it is possible to perform code audits. (Poortvliet, 2017)

4.1 The Elastic Stack overview

Elastic stack, previously known as ELK, is a group of open-source products developed by Elastic that consists of Elasticsearch, Logstash, Kibana and Beats, which together provide an end-to-end log analysis with deep searching, analysis and visualization tools. First developed separately and for different purposes, those components were merged into a single stack in 2012 (Beats were introduced in 2015, which turned ELK into the Elastic Stack). Elasticsearch is a search engine, Logstash is a log aggregator and Kibana is a visualization tool. Beats is a collection of data shipping agents. The Elastic Stack is available on-premises as well as SaaS (Software as a service) on the Elastic cloud. The on-premises solution features a few types of subscriptions: open source, basic, Gold, Platinum and Enterprise. The last three are paid and each offer extra services at an increased cost. (Elastic.co, 2019b)



Picture 5. The Elastic Stack components. (Elastic.co, 2019b)

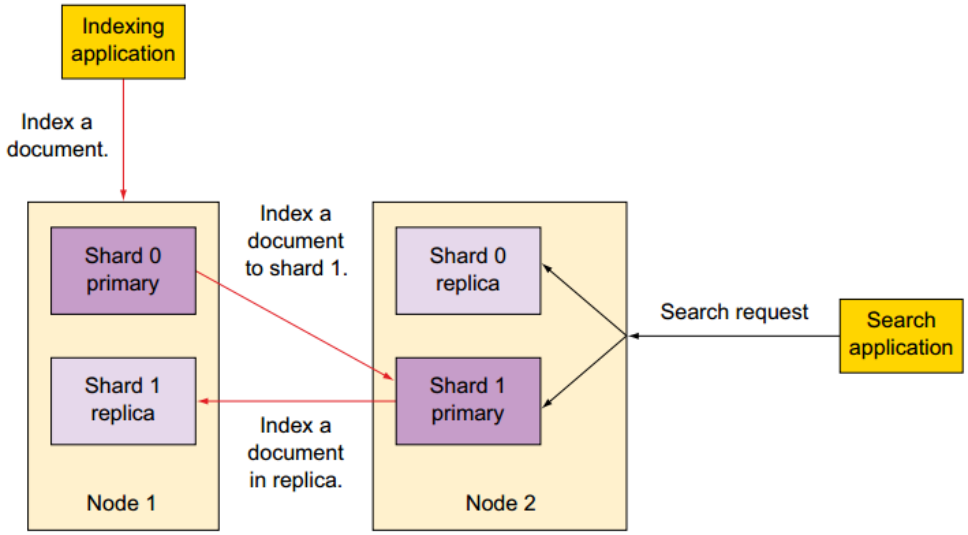
4.2 Elasticsearch

Elasticsearch is a powerful open source search and analytics engine for all types of data based on the Apache Lucene library. It features a RESTful API, is schema-free and has a web interface, and can do a full-text search and index documents in near real-time. To handle high data volume, Elasticsearch uses shards to distribute high workload. (Elastic.co, 2019c)

Every Elasticsearch instance is a cluster of one or multiple nodes. It is possible to run a single node cluster, but efficient searching and indexing capabilities of Elasticsearch come from the ability to distribute tasks across multiple nodes. Those nodes can be

assigned different tasks such as storing data and executing data-related operations like searching. Other nodes can be responsible for forwarding cluster requests or for pre-processing documents. One of those nodes is always a master node, which manages and configures all the actions in the cluster. Every node in a cluster can handle HTTP and Transport traffic and every node is aware of other nodes. (Berman, 2018)

Elasticsearch is essentially a NoSQL database: it can ingest structured or unstructured data. Elasticsearch does not use rows or columns to store information, instead, it uses serialized JSON documents. A collection of related documents is called an index. In multi-node clusters those documents are usually distributed across the whole cluster to enable immediate data access from any node. Furthermore, to protect data from hardware failures and to increase capacity, Elasticsearch uses shard replicas. Essentially an index is a group of physical shards: each document within an index has one primary shard. In Elasticsearch those primaries have their own copies, replicas. If some shards or nodes fail, Elasticsearch is still able to perform a proper search by doing so in parallel on those replicas. (Elastic.co, 2019c)



Picture 6. Elasticsearch distribution. (Techartifact, 2016)

In a typical Elastic Stack pipeline, data will be parsed with Logstash first, and then the resulting JSON is indexed into Elasticsearch. As mentioned in chapter 2, indexing is what makes these Elasticsearch documents searchable in near real-time. This speed comes from a data structure called an inverted index, which contains a list of unique words that can be found in a document as well as a list of documents for each word in which it

appears. This means when a user is searching for a specific term, it can be looked up only once, which enables a very fast full-text search. (Elastic.co, 2019c)

For indexing and searching data, Elasticsearch provides a simple REST API. To put a document into some index, the following PUT request can be used:

```
PUT /customer/_doc/1
{
  "name": "John Doe"
}
```

This request adds a new document with a unique ID of 1 and puts it into the “customer” index. If such index does not already exist, Elasticsearch creates one automatically.

To retrieve the newly created document, the following GET request can be used:

```
GET /customer/_doc/1
```

The response looks like this:

```
{
  "_index" : "customer",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 26,
  "_primary_term" : 4,
  "found" : true,
  "_source" : {
    "name": "John Doe"
  }
}
```

(Elastic.co, 2019c)

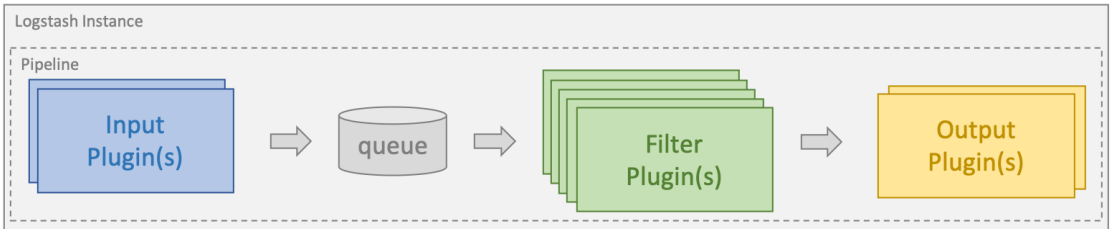
4.3 Logstash

Logstash is an open source collection of tools used for collecting, processing and forwarding data. It is based on input-plugins that enable it to ingest various kinds of data from a multitude of sources including Beats, Azure Event Hubs, Java standard input, Twitter, etc. (Elastic.co, 2019f)

When data is travelling through the Elastic stack pipeline from source to storage, it is queued in memory. Logstash is capable of performing powerful transformation and preparation regardless of the format and complexity. To do that Logstash reads queued data in small batches and uses a large number of filter plugins which are aimed at different types of data processing. (Elastic.co, 2019f)

After data processing is completed, Logstash can route data to a variety of outputs such as Elasticsearch, Google Cloud Storage, MongoDB, and standard output to name a few. It is also possible to configure a custom output using an API for plugin development.

Data processing can be handled in one or multiple pipelines (one by default). This allows to separate logical flows with different types of input and output for each pipeline. Here is an example of a single pipeline:



Picture 7. Logstash pipeline. (Dahlqvist, 2018)

The scope of this work focuses on ingesting data from Filebeat and forwarding it to Elasticsearch using Logstash. Other pipeline variations are not going to be reviewed.

To parse log data Logstash uses filter plugins such as JSON, XML, CSV and Ruby to name a few. Logstash is also capable of splitting multi-line messages into separate events, aggregating metrics data, removing special characters, extracting numbers from strings, adding geographical information from IP addresses and much more. However, log data exists in so many forms that there might not be a perfect filter for it. (Dahlqvist, 2018) For example, log messages below depict example Squid cache access data:

```
1524206424.034   19395 207.96.0.0 TCP_MISS/304 15363 GET
http://elastic.co/android-chrome-192x192.gif - DIRECT/10.0.5.120 -
1524206424.145     106 207.96.0.0 TCP_HIT/200 68247 GET
http://elastic.co/guide/en/logstash/current/images/logstash.gif - NONE/-
image/gif
```

Parsing log data in text form, such as in the example above, is possible with two common Logstash filters: dissect, which works with delimiters, and grok, which uses regular expression matching. (Dahlqvist, 2018)

The dissect filter works best if the structure of log data is consistent. Instead of matching specific patterns like in regular expressions, dissect matches delimiters, for example, spaces. Whatever is left between those delimiters becomes one of the parsed fields. (Dahlqvist, 2018)

To parse one of the earlier examples:

```
filter {
 dissect {
   mapping => {
     "message" => "%{timestamp->} %{duration} %{client_address}
%{cache_result}/%{status_code} %{bytes} %{request_method} %{url} %{user}
%{hierarchy_code}/%{server} %{content_type}"
    }
   remove_field => [“message”]
  }
}
```

The result looks like:

```
{
 "user" => "-",
 "content_type" => "-",
 "host" => "localhost",
 "cache_result" => "TCP_MISS",
 "@timestamp" => 2018-04-24T12:43:07.406Z,
 "duration" => "19395",
 "request_method" => "GET",
 "url" => "http://elastic.co/android-chrome-192x192.gif",
 "timestamp" => "1524206424.034",
 "status_code" => "304",
 "server" => "10.0.5.120",
 "@version" => "1",
 "client_address" => "207.96.0.0",
 "bytes" => "15363",
 "path" => "/home/logstash/testdata.log",
 "hierarchy_code" => "DIRECT"
}
```

Grok uses regular expressions and needs specific patterns for matching. Matching is done from left to right. There is a list of common standard patterns such as WORD, which

matches a single word, or IP, which matches an IPv4 or IPv6 IP address. GREEDYDATA matches all of the remaining data. (Dahlqvist, 2018)

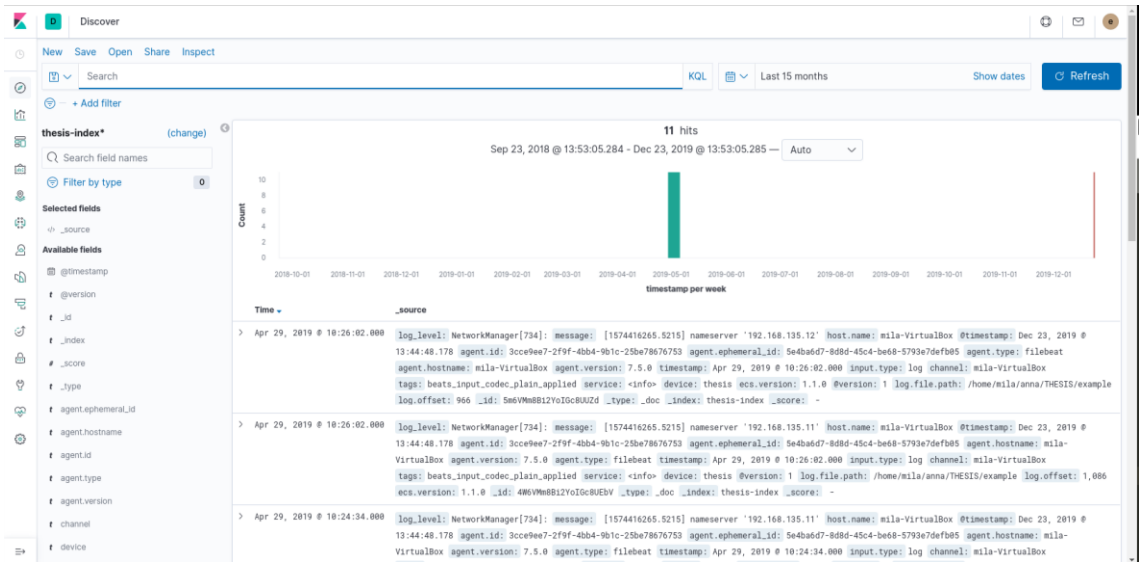
```
filter {
  grok {
    match => {
      "message" => "%{NUMBER:timestamp}%{SPACE}%{GREEDYDATA:rest}"
    }
  }
}
```

If it is possible, dissect should be preferred over grok since the latter can slow up the whole pipeline if used incorrectly. Failed matches increase performance penalties, which obviously leads to unhappy users. In fact, dissect was developed as an alternative solution for battling potential grok issues. (Boertje, 2016)

4.4 Kibana

Kibana is an open source data visualization platform built to work with Elasticsearch. Kibana is used for viewing, searching and monitoring data stored in Elasticsearch indices through a browser-based interface. On top of that, with Kibana users can get a better understanding of large volumes of data by visualizing it with pie charts, histograms, heat maps and line graphs. (Elastic.co, 2019e)

To view specific data in Kibana the index pattern needs to be selected. An index pattern defines Elasticsearch indices that contain information the user wants to access and view.



Picture 8. Kibana Discover page. (Elastic.co, 2019e)

The Discover page (Picture 8) shows documents indexed into the *thesis-index.* The number of matching documents is referred to as *hits*, which is displayed in the toolbar. By default, Kibana sorts the hits in reverse chronological order, showing the newest documents first. (Elastic.co, 2019e)

To perform a search across the indices, Kibana uses its own standard Kibana Query Language (KQL). It's simple, includes a scripted field support as well as autocomplete. It follows the Lucene key:value syntax, which uses a colon to separate fields from values.

```
response:200
```

The query above will match every document with the field "response" which has the value of 200.

```
message:"Hello World"
```

This query will search for an exact match in the message field. Skipping the quotation marks will result in breaking the phrase into separate tokens, and any document with either "Hello" or "World" in the message field will get matched.

Visualizations in Kibana are based on Elasticsearch search queries. Aggregated data that is extracted and processed can be used to create different kinds of charts, tables and maps to gain deeper insight into data. All of the pie charts, time series tools, geo maps and data tables can be combined into a dynamic custom dashboard. (Elastic.co, 2019e)

4.5 Beats and Filebeat

Beats is a collection of lightweight data shippers used for sending data from different sources and machines directly into Logstash or Elasticsearch. Beats were introduced in Elastic Stack 5.0 and consist of Filebeat, Metricbeat, Packetbeat, Winlogbeat, Auditbeat, Heartbeat, Functionbeat, each designed to ship different kinds of data. Since the scope of this work focuses on log data, only one kind of Beats shipper, Filebeat, will be reviewed. (Elastic.co, 2019a)

Before Beats and Filebeat were introduced to Elastic, Logstash was used for both handling the data transformation (including filtering and aggregation) and pulling logs from multiple sources and pushing them down the data pipeline, usually into Elasticsearch. However, with Logstash handling both data filtering and data pipelines,

the memory consumption would grow drastically and slow down the whole process. This pushed for a change and soon Elastic released Filebeat as a part of the new collection of data shippers.

Filebeat monitors locations that are specified for log files, then collects and forwards them further down the Elastic pipeline. Filebeat can ship data directly to Elasticsearch and it even provides some built-in modules that can parse common log formats such as syslog, Apache logs, MySQL, etc. Filebeat, however, does not offer capabilities for advanced log processing, so it cannot replace Logstash. (Elastic.co, 2019d)

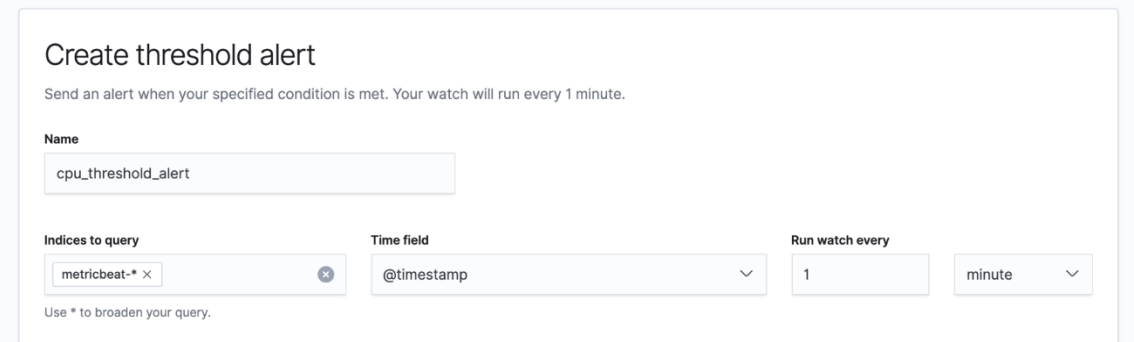
4.6 X-Pack

Originally X-Pack was developed as an extension for the Elastic Stack. It provided extra features such as security, alerting and monitoring, machine learning, etc. Most of those features were only available for the paid subscriptions, but since 2018 X-Pack has been available by default with the basic Elasticsearch installation. Some advanced security and machine learning features of X-Pack are still only available for paid subscriptions. (Kearns, 2019)

X-Pack enables role-based access control as well as TLS encrypted communication. With X-Pack it is possible to add cluster passwords for Elasticsearch, enable anonymous access, and perform message authentication as well as audit security events. (Kearns, 2019)

4.7 Alerting

On top of querying and visualization, Elasticsearch provides another option to monitor and analyze log data – alerting. Alerting comes as a part of the X-Pack. One of those alerting tools is Watcher, which can be used to create various “watches” which monitor log data periodically. (Elastic.co, 2019e)



Picture 9. Creating an alert in Watcher. (Elastic.co, 2019e)

The alerting features in X-Pack are not available for open-source and Basic subscriptions. One of the free and open-source alternatives is Elastalert. It can be configured in a separate environment from Elasticsearch and only needs to know on which port Elasticsearch is running. (Yelp, 2014)

To start alerting, Elastalert needs to be configured to run a specific rule. Possible rules include:

- *frequency* rules, where X amount of events happen within a specific time period
- *spike* rules, where the rate of events increases or decreases
- *flatline* rules, where there are less than X amount of events within a specific time period
- *blacklist* and whitelist rules, where a field matches one of the lists
- *any* rules, where an event matches a given filter
- *change* rules, where a field changes values within a specific time period

(Yelp, 2014)

Besides the rule type, each rule needs to have defined filters and what alert it sends. Filters are commonly used for partial and full matching of strings, matching integer ranges, and wildcard matching. To send alerts, Elastalert offers multiple endpoints that can be configured with Elasticsearch: e-mail, Slack, JIRA, Telegram and Amazon's SNS to name a few. (Yelp, 2014)

The example below shows how to set up a frequency rule that sends alerts to a specified Slack channel. The alert is sent if the log in process fails three times within an hour.

```
name: slack-demo
type: frequency
index: logstash-*
num_events: 3
timeframe:
    hours: 1
filter:
- query:
    query_string:
      query: "message: login failure"
alert:
- "slack"
slack:
slack_webhook_url: "<webhook>"
```

5 ELASTIC STACK IMPLEMENTATION

5.1 Prerequisites and preparations for implementation

There are a few recommendations that need to be considered before setting up the Elastic stack for deployment: memory allocation, CPU usage, and disk space management.

5.1.1 Memory

Memory is a crucial part of the stack and it is very likely it will be one of the first resources to run out.

Elasticsearch runs on JVM and many of its data structures that are based on Lucene are disk-based formats, which are too large to be loaded into main memory. Hence, it is important that there is enough heap space available for Elasticsearch. The heap size is specified under JVM options and determined depending on the amount of available RAM: no more than half of physical RAM used. Although there are no hard and fast rules regarding these values, the Elastic stack engineers recommend to keep the heap size around 30 to 32 gigabytes. (Elastic.co, 2019c)

Too small heap size can result in memory errors and the application won't be able to handle multiple applications simultaneously. Java is a garbage-collected language, which means it occasionally has garbage-collection pauses. Those pauses are initiated when some region of memory is full and Java needs to remove data that is no longer needed. During these pauses JVM suspends other operations. This will largely affect the end-user experience as Elasticsearch will reduce the amount of queries and indexing per second that it will be able to perform during these frequent and short pauses. Configuring too large of a heap size is no good either: with a larger heap size, JVM will have infrequent and long pauses, which Elasticsearch is not able to differentiate from a node that is unreachable. When a node hangs, Elasticsearch removes it from the cluster and reallocated its shards. This makes the cluster unstable. (Tedor, 2016)

5.1.2 CPUs

According to the Elastic Stack guide, most Elasticsearch deployments do not require extensive CPU resources. High CPU usage is expected when indexing and querying big

volumes of data, but only for a short period of time. These spikes in CPU usage can occur in JVM if the heap is too small. (Elastic.co, 2019f)

The type of processor does not matter as much as proper configuration of other resources such as memory and disk space.

5.1.3 Disk space

As mentioned in chapter 4, the power of Elasticsearch comes from its ability to distribute tasks across multiple nodes in the cluster. While it is possible to run a functional cluster of one node, adding more nodes increases the capacity and reliability of the cluster. When there is only one node, there are no replica shards and all primary shards are located on the single node. If something happens to this node, data is at risk. However, when the cluster has more than one node, it automatically creates and allocates replica shards. (Elastic.co, 2019c)

Before Elasticsearch allocates new shards (or relocates old ones) to a node in the cluster, it looks at the disk space available on that node. There are default values that control shard allocation. By default, if the node has used up more than 85% of disk space, Elasticsearch won't allocate shards to it. (Elastic.co, 2019c)

It is also important to keep in mind that the size of raw data will most likely change once it is transformed during indexing. This differs for different types of data and how it was enriched. For example, indexing logs that are in JSON format without adding any additional data to it probably won't change the size. However, if the data is initially unstructured and needs additional information before being indexed, its size will most likely increase. There is no single way to calculate an exact amount of storage required for indexing data, but one option is to avoid testing with small amount of data before going into production. (Dahlqvist, 2018)

A similar approach should be used with estimating how much querying will take up disk space. To estimate this, the best option is to simulate the levels of querying that will be done in production.

According to a blog on Elasticsearch storage requirements (Dahlqvist, 2017), there are some ways to optimize on-disk storage. One is to avoid using default settings in Filebeat that enable dynamic field mapping. Dynamic field mapping refers to labelling all fields in a document as text in order to allow free text search. With custom mappings it is possible

to label fields that do not require text-search as keywords. This small optimization can save up to 20% of disk space. (Dahlqvist, 2017)

Another solution is to remove unnecessary fields in a document before indexing. When raw data is indexed into Elasticsearch, it goes through enrichment with new fields being added to the log. (Dahlqvist, 2017)

```
Jan  5 19:49:15 mila-VirtualBox anacron[845]: Normal exit (1 job run)
```

The syslog message from above will look like this after Filebeat sends to Logstash where it is parsed:

```
{
           "ecs" => {
       "version" => "1.1.0"
    },
      "@version" => "1",
           "log" => {
        "offset" => 70519,
          "file" => {
            "path" => "/var/log/syslog"
        }
    },
          "host" => {
        "name" => "mila-VirtualBox"
    },
     "timestamp" => "2020-01-05T19:49:15+03:00",
     "program" => "anacron[845]",
         "input" => {
        "type" => "log"
    },
      "hostname" => "mila-VirtualBox",
         "agent" => {
            "hostname" => "mila-VirtualBox",
                  "id" => "3cce9ee7-2f9f-4bb4-9b1c-25be78676753",
             "version" => "7.5.0",
                "type" => "filebeat",
        "ephemeral_id" => "60444a6a-060c-4c15-8a0f-0f669a5eab8c"
    },
    "@timestamp" => 2020-01-05T20:21:57.387Z,
       "message" => "Normal exit (1 job run)",
          "tags" => [
        [0] "beats_input_codec_plain_applied"
    ]
```

The highlighted bold and orange fields were added after enrichment. While this new data provides additional information about the log files that can be useful to the user, it also takes up more space. If some of the fields are defined to be unnecessary, it can be

beneficial to remove them to save some storage space. Depending on how many fields are removed, up to 17% of disk space can be saved. (Dahlqvist, 2017)

Another important consideration is data compression, which can have a big impact on disk space. By default, Elasticsearch compresses data before writing it to a disk using a specific algorithm, which can be swapped for another, more aggressive alternative. It comes with a trade-off, which affects the overall indexing performance by using more CPU. It might be a good option in the long run since it saves up a significant amount of storage. (Dahlqvist, 2017)

5.2 Implementation

The implementation in the scope of this study is lightweight and is carried out for the purpose of testing out basic features of the Elastic Stack. The stack was set up on a single server in a virtual environment running Ubuntu 18.04 with Intel Xeon 2 vCPUs and 8 gigabytes of RAM.

Following the memory allocation practices defined in an earlier chapter, Elasticsearch should be assigned roughly half of the memory available on the server. Eight gigabytes of RAM is definitely not enough for large deployments unless there is a large quantity of those small 8-gigabyte machines. Since this implementation was only used to test the basics of the Elastic stack, 8 gigabytes was deemed enough and did not affect overall performance. However, for deploying Elasticsearch to production, the recommended amount of RAM is 64 gigabytes. It is important to remember that Logstash and Kibana should also be accounted when allocating memory in the stack.

To test one of the optimization tricks mentioned earlier in this chapter, a sample of raw log data was ingested into Elasticsearch. The same sample was ingested into a different index, but with reduced field count. The Filebeat fields that are bolded red below were removed from each log file:

```
"hostname" => "mila-VirtualBox",
  "agent" => {
     "hostname" => "mila-VirtualBox",
           "id" => "3cce9ee7-2f9f-4bb4-9b1c-25be78676753",
      "version" => "7.5.0",
         "type" => "filebeat",
 "ephemeral_id" => "60444a6a-060c-4c15-8a0f-0f669a5eab8c"
```

The table below compares the sizes of the two samples:

Table 1. Results of the optimization test.

	Raw data size (MB)	Enriched and indexed data size (MB)	Size compared to raw data (%)
Original	50.2	69.3	138%
With reduced field count	50.2	66.2	132%

In this case the amount of disk space saved from removing unnecessary fields is 6%. It may seem small, but it might be worthy in the long run, especially if combined with other methods of disk optimization such an aggressive compression algorithm or disabling the default dynamic mapping.

As mentioned previously, Elasticsearch does not require extensive CPU resources. During implementation it was observed that the average CPU usage during three weeks of retention period was around 6%. The high spikes when the CPU usage exceeded 60% were only observed briefly during Logstash start-up and data ingestion. It did not affect any of the ongoing processes in the rest of the stack.

5.2.1 Security

Before merging into production, the server responsible for hosting the Elastic stack should have its network settings properly configured since data attacks can come from the network. All the traffic coming to, from and within an Elasticsearch cluster should be encrypted with SSL/TLS. (Elastic.co, 2019c)

Using X-Pack security features, role-based authentication was enabled in the cluster to prevent unauthorized access. Since the cluster was not listening on an external interface, no additional security measures were taken.

With default settings, Elasticsearch assumes that the cluster is in development mode. Hence, it is crucial that security settings are configured before running Elasticsearch. Once the network host is configured, Elasticsearch assumes the cluster is being moved to production. (Elastic.co, 2019c)

5.2.2 Log processing

As mentioned in Chapter 3, the raw log files that are used for this implementation contain erroneous timestamps, which need to be fixed during data preprocessing. Since the data in these log files is unstructured, a custom solution needed to be implemented to handle this issue. Appendix 1 contains a script written in Python that tackles timestamp fixing.

Once the data is preprocessed, it is ready to be ingested into Elasticsearch. Ideally, each device would send its own log files to Elasticsearch, but since log collection is outside of the scope of this study, a custom file location was set up where log files were sorted by device serial numbers. When Filebeat is running, it automatically detects new files added to the file location and forwards them to Logstash.

Logstash parses the log data with a simple configuration file:

```
input {
    beats {
        port => "5044"
    }
}
# The filter part of this file is commented out to indicate that it is
# optional.
filter {
    dissect {
      mapping => {
        "message" => "%{log_timestamp} %{channel} %{log_level} %{service}
%{message}"
      }
    }
    grok {
      match => { "[log][file][path]" =>
"/home/mila/anna/%{GREEDYDATA:device}/" }
      break_on_match => false
    }
    mutate {
      lowercase => [ "device" ]
    }
}
output {
  elasticsearch {
      hosts => [ "localhost:9200" ]
      index => "%{device}-index"
  }
}
```

Logstash uses the log file path to determine which device it is coming from and to index into the correct index. In the future, the configuration file might need to be tweaked depending on how log collection is implemented.

Now the log file that is indexed into Elasticsearch contains two timestamps: *@timestamp*, which is added by Filebeat during the enrichment process, and *log_timestamp*, which is

added by Logstash. Choosing to keep two timestamps can be helpful in case the timestamp fixing fails at any point during data preprocessing. This way the user will still be able to determine an approximate time around which the events in the log files took place.

5.2 Potential issues and drawbacks

The current implementation of the Elastic Stack works exactly how it was intended to: log files are shipped into Logstash, where they are parsed and processed. Afterwards they are ingested into specific indices into Elasticsearch. Kibana can be used to create different types of visualizations, and ElastAlert can be configured to send various kind of alerts. Its initial setup is relatively simple due to the low volume of data that is being ingested, but what happens when the size of the cluster keep growing?

Bigger cluster means higher complexity. While the current implementation of the stack is able to handle log collection, this might not be the case when the amount of log files being ingested into Elasticsearch grows significantly. When the stack is pushed to production, it is most likely that there will be a need for multiple instances of Logstash to handle different data sources. Logstash should also be configured to handle data loss that comes with high spikes in data volume. A queuing and buffer mechanism should be configured to tackle this issue, for example, Kafka or Redis.

Another important consideration when designing the Elastic Stack solution is capacity planning. Earlier in this chapter, it was identified that memory management is crucial when setting up the Elastic Stack to ensure that it continues performing at the same speed after scaling up. This will require serious commitment and a proper long-term storage strategy as the system scales. Additionally, to perform at peak capacity, the system will require the right kind of hardware. Buying additional SSD disks can already be quite expensive. Having mentioned that, unfortunately, complexity is not the only factor that grows as the Elastic Stack implementation scales up and requires more configuration and fine-tuning. Even though this varies for each use case, configuring additional features can financially cripple an organization that is tight on a budget.

Security is another place where costs can add up. As mentioned earlier, proper security configuration is crucial before merging the stack into production. While essential features such as role-based authentication can be configured with the basic subscription X-Pack,

several of its advanced tools are still not available for those tiers: IP filtering, audit logging, token services, field- and document-level security to name a few. Hence, it is very important to identify what kind of security level is expected from the log management system before making a decision whether or not the free subscriptions of the Elastic stack fulfill the criteria. The same planning should take place when considering machine learning, monitoring and alerting tools that are only available for the commercial versions of the Elastic Stack.

All of the examples mentioned above show that setting up a highly available environment with multiple clusters will not be easy and will consume time and resources. The performance of the environment solely depends on how well the Elastic Stack infrastructure is organized to scale from the beginning. More costs add up as the need for additional storage grows. Enabling advanced security features as well as intelligence and analysis tools does not come with free subscriptions. Without commitment, expertise and serious capital investment, maintaining a powerful and highly available Elastic Stack cluster can become a tedious task with significant financial costs.

6 CONCLUSION

Setting up a log management solution is no easy task: before choosing tools and services for building such solution it is important to look at the data before it is stored for analysis. The vast amount of raw data that is constantly being produced results in more occurrences of inconsistencies and errors within the data. High quality data ensures better predictions and results, thus, data preprocessing plays a vital role in data analysis.

Data preprocessing can be performed using various tools. This thesis focused on how to build a custom prototype with Python to fix incorrect timestamps in log files.

The implementation of the Elastic Stack fulfills the basic functionalities of a log management solution. Filebeat is used to collect logs and can be configured to perform simple parsing. Logstash transforms data and prepares it for further ingestion. Elasticsearch stores the data and provides tools for querying. Kibana offers users visual feedback and analysis of the data. In addition to the main components of the stack, open source alternatives such as ElastAlert can be used to configure alerting for specified indices.

In order to reap all the benefits of the Elastic Stack, the current implementation needs to be improved in the future. Additional security measures need to be taken before the stack is merged into production to ensure that the client data is not compromised. Log data that contains personal and sensitive information should also be considered to be anonymized before being ingested into Elasticsearch. Additional dashboards and visualizations in Kibana must be set up in order to perform deeper and more detailed log analysis.

The findings of this study support that the Elastic Stack is a reasonable choice for a log management solution. When choosing between the Elastic Stack and a commercial solution, it should be noted that the former requires more labor and financial resources if the implementation needs to be scaled up. Adding additional features and storage to the stack can take its toll on a tight budget.

REFERENCES

Berman, D., 2018. *Creating an Elasticsearch Cluster: Getting Started.* [Online]
Available at: https://logz.io/blog/elasticsearch-cluster-tutorial/
[Accessed 1 December 2019].

Boertje, G., 2016. *Logstash Dude, where's my chainsaw? I need to dissect my logs.* [Online]
Available at: https://www.elastic.co/blog/logstash-dude-wheres-my-chainsaw-i-need-to-dissect-my-logs
[Accessed 3 December 2019].

Chuvakin, A., 2010. *The Complete Guide to Log and Event Management,* s.l.: s.n.

Dahlqvist, C., 2017. *Filebeat modules, access logs and Elasticsearch storage requirements.* [Online]
Available at: https://www.elastic.co/blog/filebeat-modiles-access-logs-and-elasticsearch-storage-requirements
[Accessed 30 December 2019].

Dahlqvist, C., 2018. *A Practical Introduction to Logstash.* [Online]
Available at: https://www.elastic.co/blog/a-practical-introduction-to-logstash
[Accessed 2 December 2019].

Dahlqvist, C., 2018. *Sizing Hot-Warm Architectures for Logging and Metric in the Elasticsearch Service on Elastic Cloud.* [Online]
Available at: https://www.elastic.co/blog/sizing-hot-warm-architectures-for-logging-and-metrics-in-the-elasticsearch-service-on-elastic-cloud
[Accessed 22 December 2019].

Elastic.co, 2019a. *Beats overview.* [Online]
Available at: https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html
[Accessed 3 December 2019].

Elastic.co, 2019b. *Elastic Stack Overview.* [Online]
Available at: https://www.elastic.co/guide/en/elastic-stack-overview/current/introduction.html
[Accessed 22 November 2019b].

Elastic.co, 2019c. *Elasticsearch Reference.* [Online]
Available at: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
[Accessed 1 December 2019c].

Elastic.co, 2019d. *Filebeat Reference.* [Online]
Available at: https://www.elastic.co/guide/en/beats/filebeat/current/index.html
[Accessed 2 December 2019d].

Elastic.co, 2019e. *Kibana Guide [7.5].* [Online]
Available at: https://www.elastic.co/guide/en/kibana/current/index.html
[Accessed 1 December 2019e].

Elastic.co, 2019f. *Logstash Reference.* [Online]
Available at: https://www.elastic.co/guide/en/logstash/7.5/index.html
[Accessed 2 December 2019f].

Evondos, 2019. *Data Security in Evondos Platform.* s.l.:s.n.

Evondos, 2019. *Service description.* [Online]
Available at: https://www.evondos.com/
[Accessed 1 November 2019].

International Organization for Standardization, n.d. *ISO 8601 Date and Time Format.* [Online]
Available at: https://www.iso.org/iso-8601-date-and-time-format.html
[Accessed 13 November 2019].

Kearns, S., 2019. *Security for Elasticsearch is now free.* [Online]
Available at: https://www.elastic.co/blog/security-for-elasticsearch-is-now-free
[Accessed 3 December 2019].

Kent, K. & Murugiah, S., 2006. *Guide to Computer Security Log Management,* Gaithersburg: National Institute of Standards and Technology.

La rosa, A., 2018. *Log Monitoring: not the ugly sister.* [Online]
Available at: https://web.archive.org/web/20180214153657/https://blog.pandorafms.org/log-monitoring/
[Accessed 11 November 2019].

Mushtaq, S., 2019. *Data preprocessing in detail.* [Online]
Available at: https://developer.ibm.com/articles/data-preprocessing-in-detail/#
[Accessed 1 December 2019].

Nguyen, T., 2018. *How to Graph Logs & Visualize Data for Proper Log Analysis.* [Online]
Available at: https://logdna.com/how-to-visualize-your-log-data/
[Accessed 11 November 2019].

Poortvliet, J., 2017. *6 reasons open source is good for business.* [Online]
Available at: https://opensource.com/article/17/10/6-reasons-choose-open-source-software
[Accessed 22 November 2019].

Python Software Foundation, 2019. *Python 3.5.9 documentation.* [Online]
Available at: https://docs.python.org/3.5/index.html
[Accessed 15 November 2019].

Sharma, M., 2018. *What Steps should one take while doing Data Preprocessing?.* [Online]
Available at: https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa
[Accessed 14 November 2019].

Shields, I., 2017. *IBM Developer.* [Online]
Available at: https://developer.ibm.com/tutorials/l-lpic1-108-2/
[Accessed 10 November 2019].

Solarwinds Loggly, 2019. *Alerting & Monitoring.* [Online]
Available at: https://www.loggly.com/docs/alerts-overview/
[Accessed 1 December 2019].

Solarwinds Loggly, n.d. *Ultimate Guide to Logging.* [Online]
Available at: https://www.loggly.com/ultimate-guide/apache-logging-basics/
[Accessed 11 November 2019].

Sumo Logic, n.d. *Log Analysis.* [Online]
Available at: https://www.sumologic.com/glossary/log-analysis/
[Accessed 11 November 2019].

Techartifact, 2016. *Shards and Replicas in Elasticsearch.* [Online]
Available at: https://www.techartifact.com/blogs/tag/elastic
[Accessed 1 December 2019].

Tedor, J., 2016. *A Heap of Trouble: Managing Elasticsearch's Managed Heap.* [Online]
Available at: https://www.elastic.co/blog/a-heap-of-trouble
[Accessed 21 December 2019].

Yelp, 2014. *ElastAlert - Easy & Flexible Alerting with Elasticsearch.* [Online]
Available at: https://elastalert.readthedocs.io/en/latest/index.html
[Accessed 2 January 2020].

Yigal, A., 2017. *Splunk and ELK Stack: A Side-by-side Comparison.* [Online] Available at: https://devops.com/splunk-elk-stack-side-side-comparison/ [Accessed 22 November 2019].

Source code for data preprocessing

```
#!/usr/bin/python
#coding=utf-8
"""
File:         fix_timestamps.py
Author:       Mila Grigoryeva
Language:     Python 3.5.2
Description:  Script for fixing timestamps and converting to ISO format in log
files
*********************************************************************
"""

import datetime
import linecache
import os
import re
import pytz

START_BOOT = 'Booting Linux on physical CPU'
END_BOOT = 'Setting system time'

PATH = ("/home/mila/thesis/")

def iso_timezone(timestamp, d_timezone):
    """Set correct timezone"""
    timezone = pytz.timezone("Europe/Helsinki")
    # For now only consider Nordic timezones
    if d_timezone == 1:
        timezone = pytz.timezone("Europe/Stockholm")
    aware_timestamp = timezone.localize(timestamp)
    iso_timestamp = aware_timestamp.isoformat()

    return iso_timestamp

def format_time(line, year, timezone):
    """Format timestamp to ISO with UTC offset and add year"""
    date_match = re.search(r'\w\w\w \s?\d{1,2} ', line)
    time_match = re.search(r'\d{2}:\d{2}:\d{2}', line)
    raw_timestamp = date_match.group() + year + " " + time_match.group()
    timestamp = datetime.datetime.strptime(raw_timestamp, '%b %d %Y %H:%M:%S')
    formatted_timestamp = iso_timezone(timestamp, timezone)

    return formatted_timestamp

def get_correct_time(log_file, correct_index):
    """Get correct time in case log file contains wrong timestamps"""
    timestamp = None
    minutes_passed = None

    # The line that follows the last line before boot ends, contains correct
time
    correct_time_line = linecache.getline(log_file, correct_index)
    minutes_passed = re.search(r'\d{2}:\d{2}:\d{2}',
correct_time_line).group()
    timestamp = re.search(r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}',
correct_time_line).group()
    time = re.search(r'\d{2}:\d{2}:\d{2}', timestamp).group()

    return time, minutes_passed

def create_time(line, correct_time_str, minutes_str, timezone):
    """Extrapolate new timestamps based on the next known correct time"""
```

```
    correct_time = datetime.datetime.strptime(correct_time_str, "%Y-%m-%d
%H:%M:%S")
    minutes = datetime.datetime.strptime(minutes_str,
"%H:%M:%S").replace(hour=0)
    old_time_str = re.search(r'\d{2}:\d{2}:\d{2}', line).group()
    old_time = datetime.datetime.strptime(old_time_str,
"%H:%M:%S").replace(hour=0)
    new_time = correct_time - minutes + old_time
    iso_new_time = iso_timezone(new_time, timezone)

    return iso_new_time

def replace_time(line, new_time):
    """Replace formatted and correct timestamps in the logfile"""
    new_line = re.sub(r'\w\w\w \s?\d{1,2} \d{2}:\d{2}:\d{2}', new_time, line)
    return new_line

def main():
    """Convert to correct format and fix timestamps"""
    wrong_time = False
    start_index = []
    end_index = []
    wrong_lines = None
    result = []
    new_line = None

    dirs = os.listdir(PATH)

    logfiles = [logfile for logfile in dirs if os.path.isfile(logfile)]
    for logfile in logfiles:
        with open(logfile) as log:
            for index, line in enumerate(log, start=1):
                if re.search(START_BOOT, line):
                    # Line contains wrong timestamp
                    wrong_time = True
                    # Write lines with wrong time to the list
                    start_index.append(index)
                elif re.search(END_BOOT, line):
                    # Sometimes setting system time line occurs when device
wakes up from sleep
                    if wrong_time:
                        wrong_time = False
                        end_index.append(index)
        # Indices of the lines that contain wrong timestamps
        wrong_lines = zip(start_index, end_index)

        new_file = open('New-{}'.format(logfile), 'a')
        with open(logfile) as log_a:
            for index, line in enumerate(log_a, start=1):
                new_time = format_time(line, "2019", 0)
                new_line = replace_time(line, new_time)

                # Fix lines with wrong timestamps
                for ranges in wrong_lines:
                    if index in range(ranges[0] - 2, ranges[1] + 1):
                        result = get_correct_time(logfile, ranges[1])
                        new_time = create_time(line, result[0], result[1], 0)
                        new_line = replace_time(line, new_time)
                new_file.write(new_line)
        new_file.close()

if __name__ == "__main__":
    main()
```